# Minimum Spanning Tree (MST)
# MST Kruskal’s Algorithm
# MST Prim’s Algorithm

**Week-11, Lecture-01**

**Course Code:** CSE221

**Course Title:** Algorithms

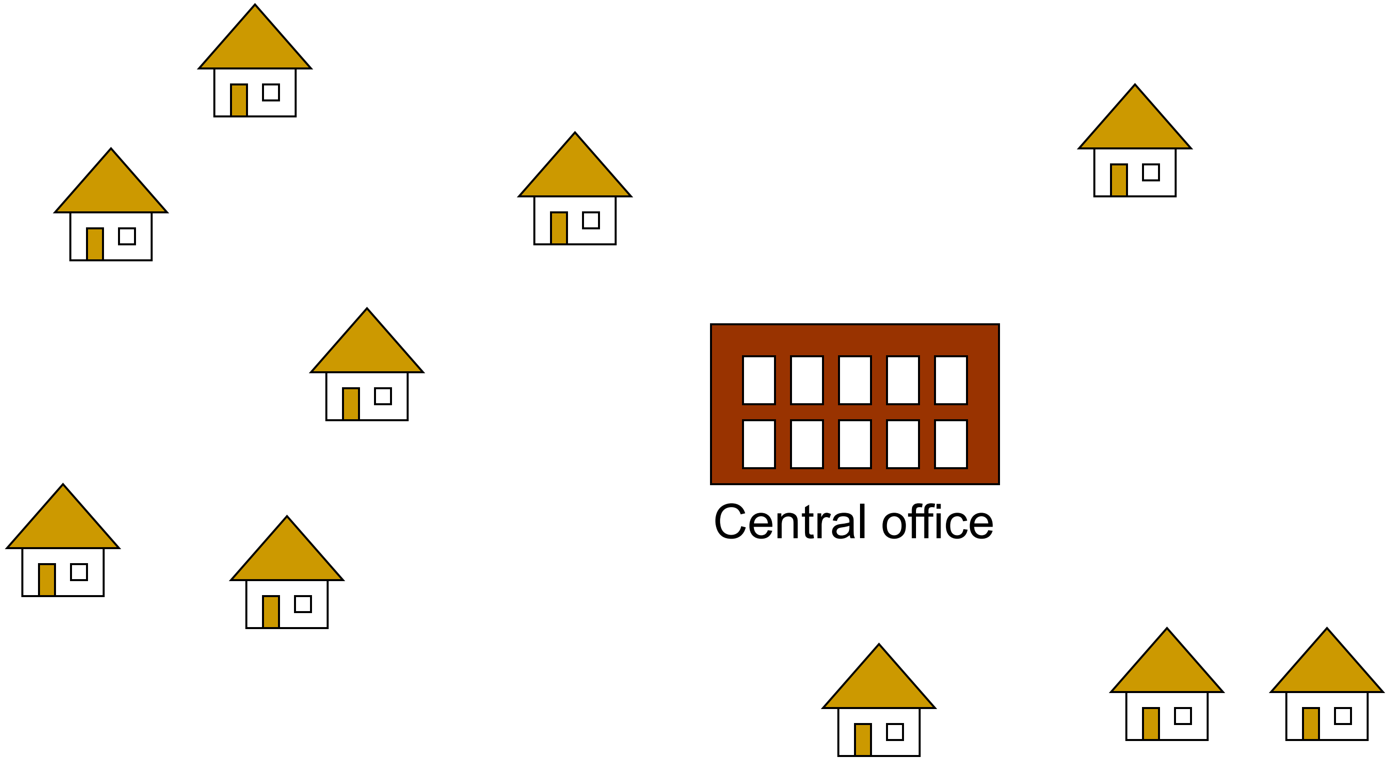
**Program:** B.Sc. in CSE

**Course Teacher:** Tanzina Afroz Rimi

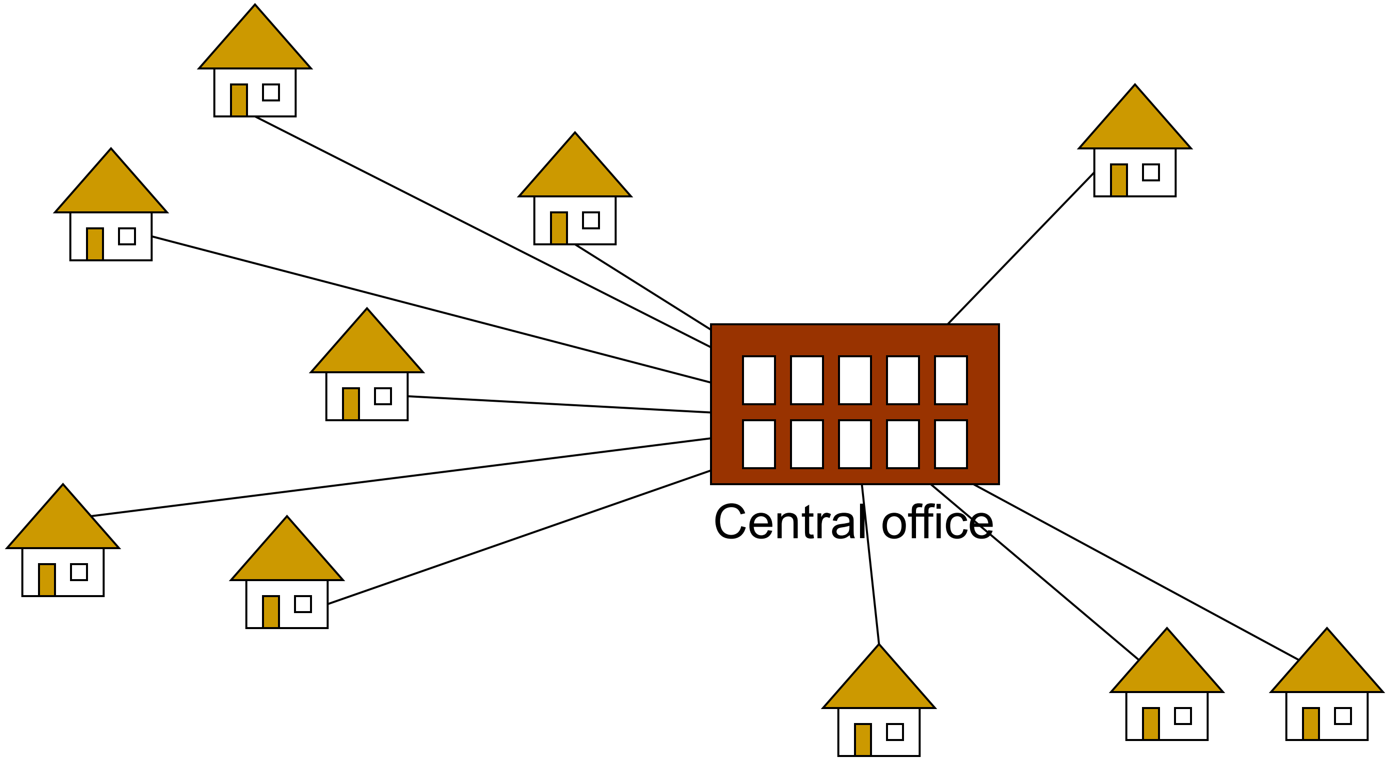
**Designation:** Lecturer

**Email:** tanzinaafroz.cse@diu.edu.bd

# Problem: Laying Telephone Wire

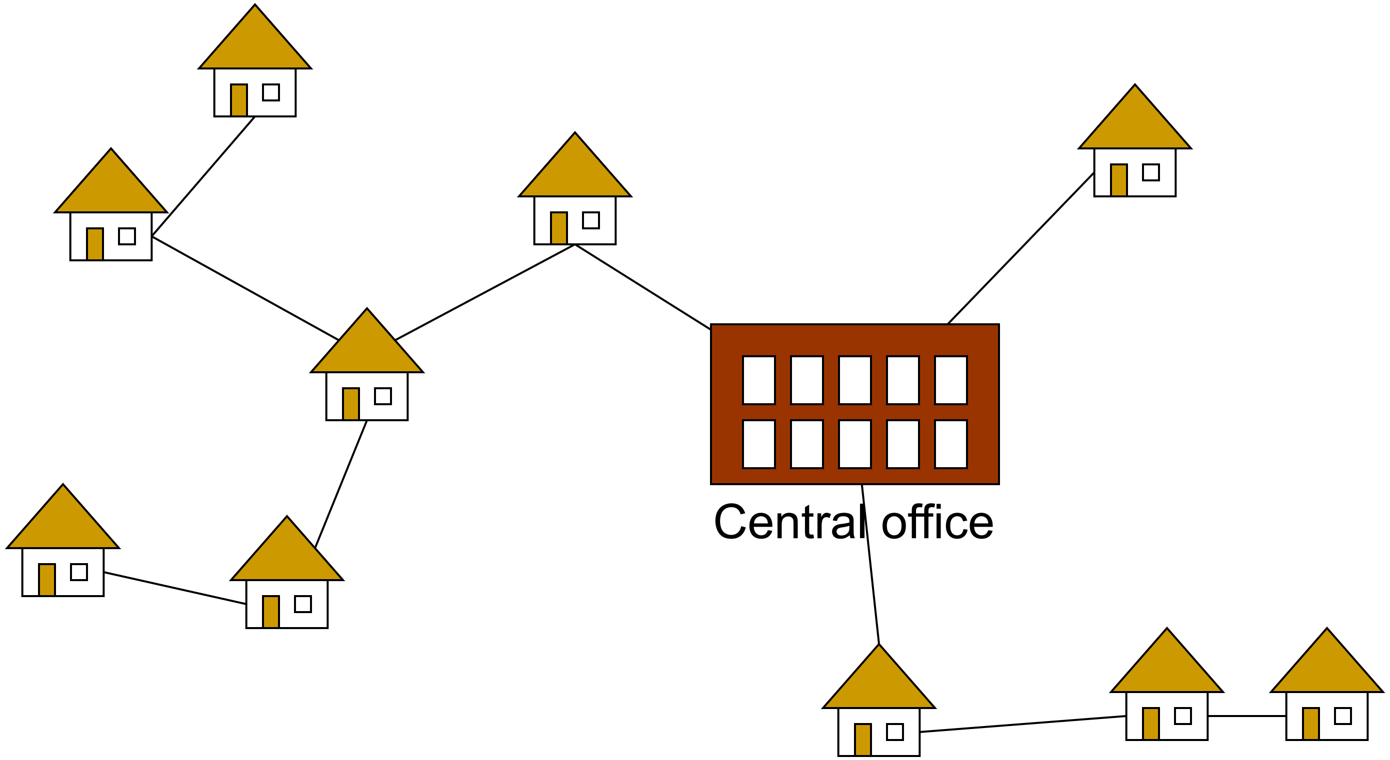


# Wiring: Naïve Approach



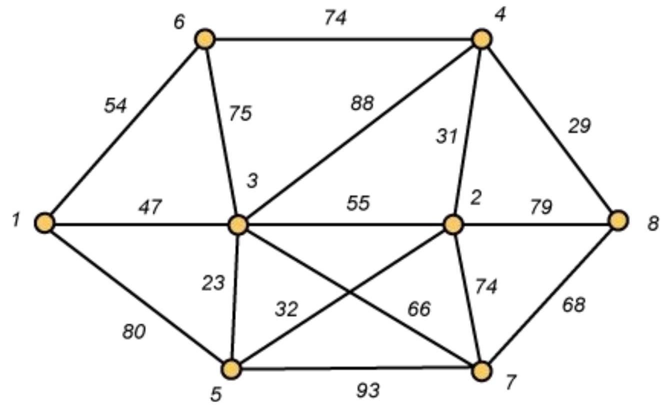

**Expensive!**

# Wiring: Better Approach




Minimize the total length of wire connecting the customers

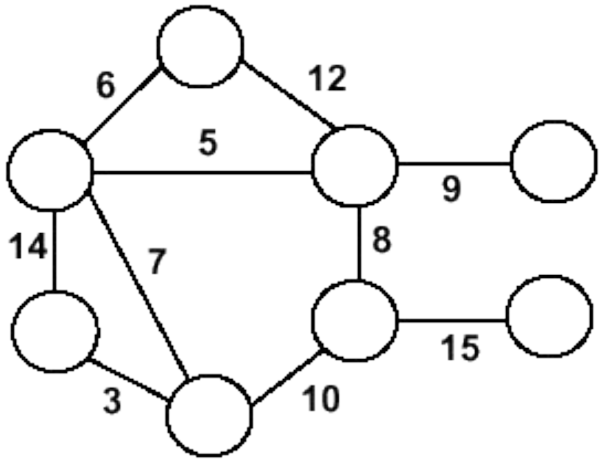
# A Networking Problem




Problem: The vertices represent 8 regional data centers which need to be connected with high-speed data lines. Feasibility studies show that the links illustrated above are possible, and the cost in millions of dollars is shown next to the link. Which links should be constructed to enable full communication (with relays allowed) and keep the total cost minimal.

# Minimum Spanning Trees

- Undirected, connected graph $G = (V,E)$
- Weight function $W: E \rightarrow R$




- Spanning tree: tree that connects all the vertices (above?)
- Minimum spanning tree: tree that connects all the vertices and minimizes

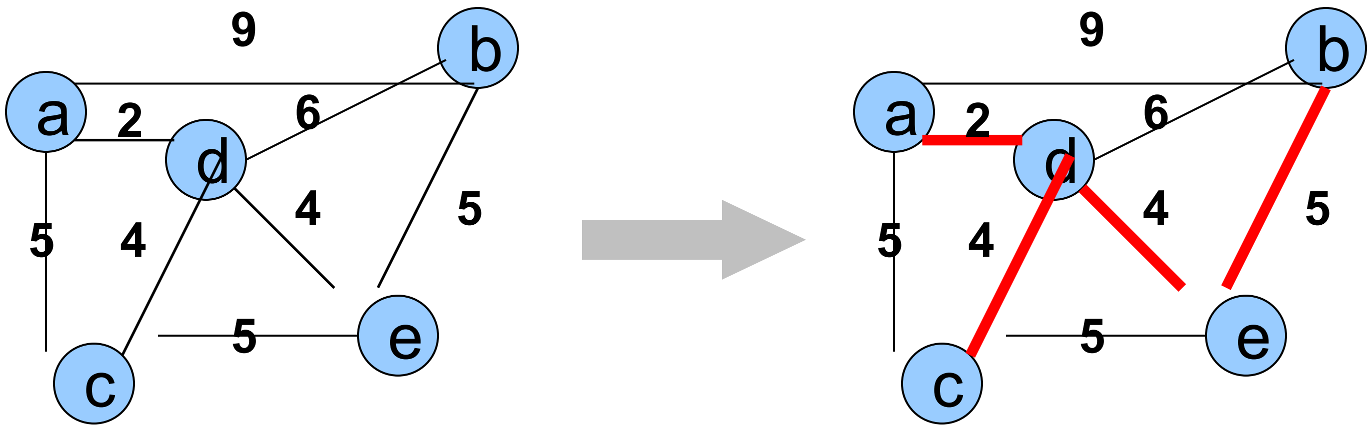
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

# Minimum Spanning Tree (MST)

A **minimum spanning tree** is a subgraph of an undirected weighted graph $G$, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices $V$
  - contains $|V| - 1$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

# How Can We Generate a MST?



# Greedy Choice

We will show two ways to build a minimum spanning tree.

- A MST can be grown from the current spanning tree by adding the nearest vertex and the edge connecting the nearest vertex to the MST. (Prim's algorithm)

- A MST can be grown from a forest of spanning trees by adding the smallest edge connecting two spanning trees. (Kruskal's algorithm)

# Notation

- Tree-vertices: in the tree constructed so far
- Non-tree vertices: rest of vertices

# Prim's Selection rule

- Select the minimum weight edge between a tree-node and a non-tree node and add to the tree

# The Prim algorithm Main Idea

**Select a vertex to be a tree-node**

**while (there are non-tree vertices) {**

**if there is no edge connecting a tree node with a non-tree node**

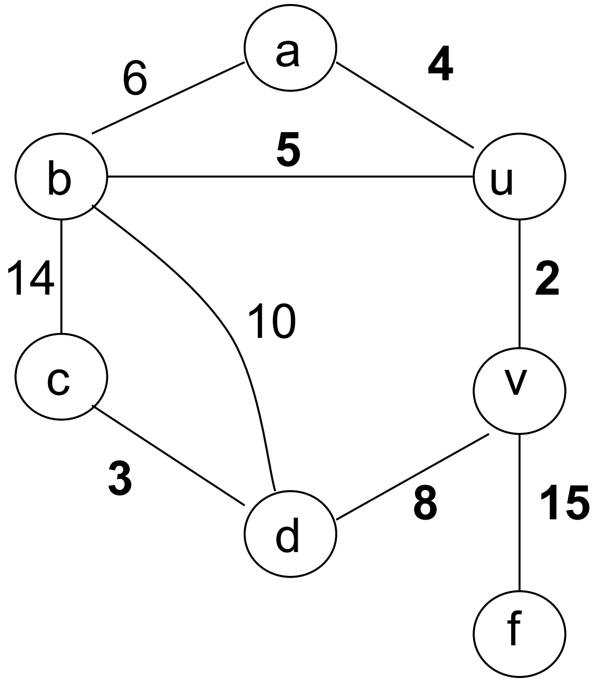
**return "no spanning tree"**

**select an edge of minimum weight between a tree node and a non-tree node**

**add the selected edge and its new vertex to the tree**

**}**

**return tree**



# Prim’s Algorithm

- Vertex based algorithm
- Grows one tree T, **one vertex at a time**

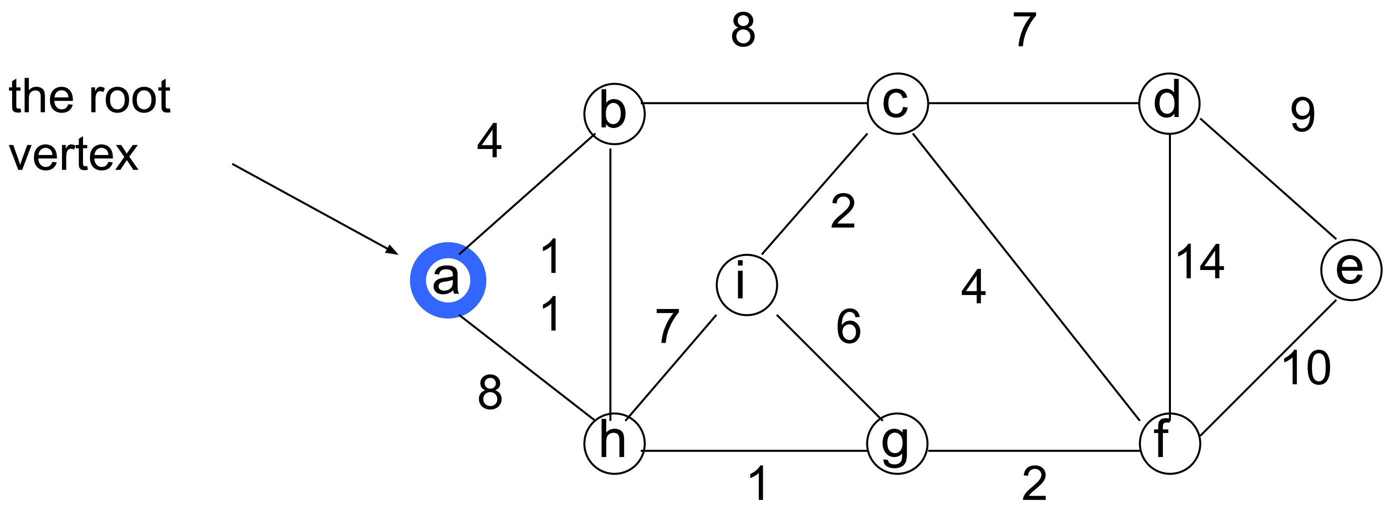
# Prim Algorithm (2)

```
MST-Prim(G,w,r)
Q ← V[G]  // Q – vertices out of T
for each u ∈ Q
   key[u] ← ∞
key[r] ← 0                 // r is the first tree node, let r=1
π[r] ← NIL
while Q ≠ ∅ do
  u ← ExtractMin(Q)  // making u part of T
     for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)
```

# Prim Algorithm:Variables

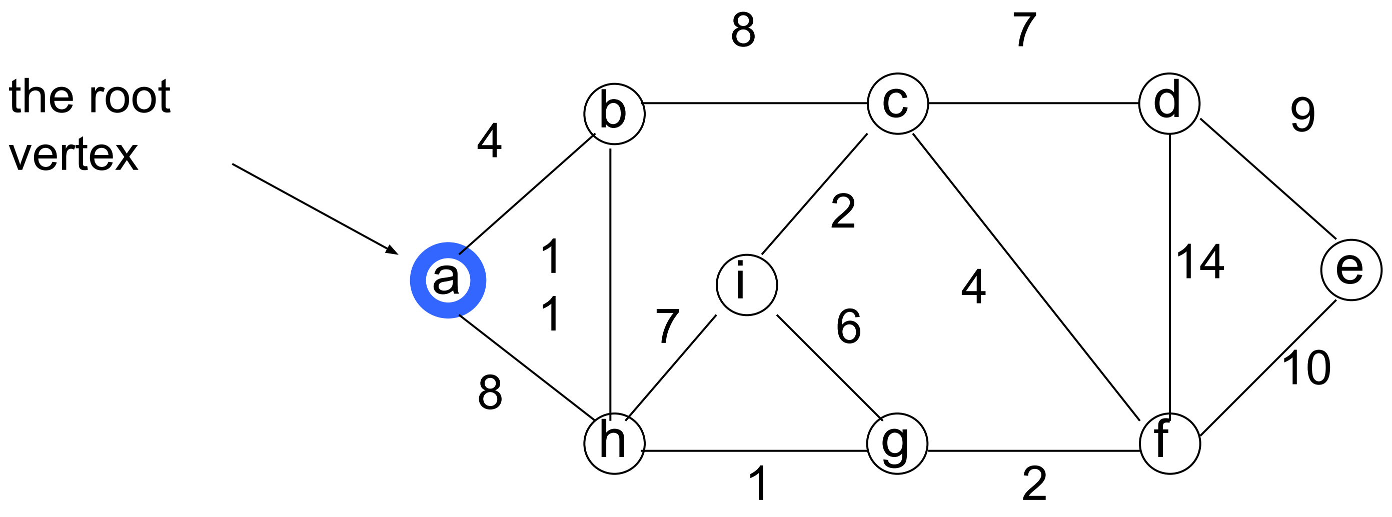
- r:
  - Grow the minimum spanning tree from the **root vertex “r”**.
- Q:
  - is a priority queue, holding all vertices that are not in the tree now.
- key[v]:
  - is the minimum weight of any edge connecting v to a vertex in the tree.
- $\pi$ [v]:
  - names the parent of v in the tree.
- T[v] –
  - Vertex v is already included in MST if T[v]==1, otherwise, it is not included yet.

# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Key | 0 | - | - | - | - | - | - | - | - |
| π | -1 | - | - | - | - | - | - | - | - |

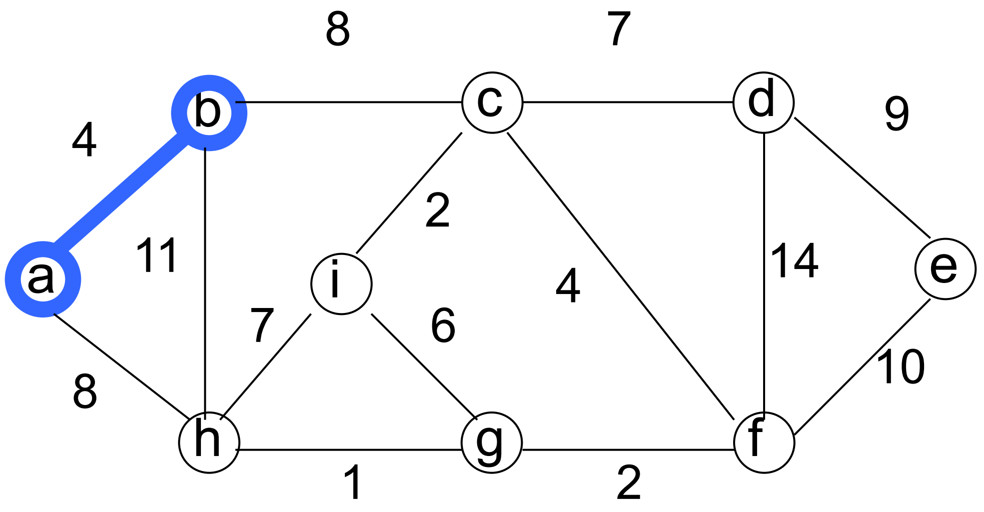
# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Key | 0 | **4** | - | - | - | - | - | 8 | - |
| π | -1 | **a** | - | - | - | - | - | a | - |

# The execution of Prim's algorithm(moderate part)

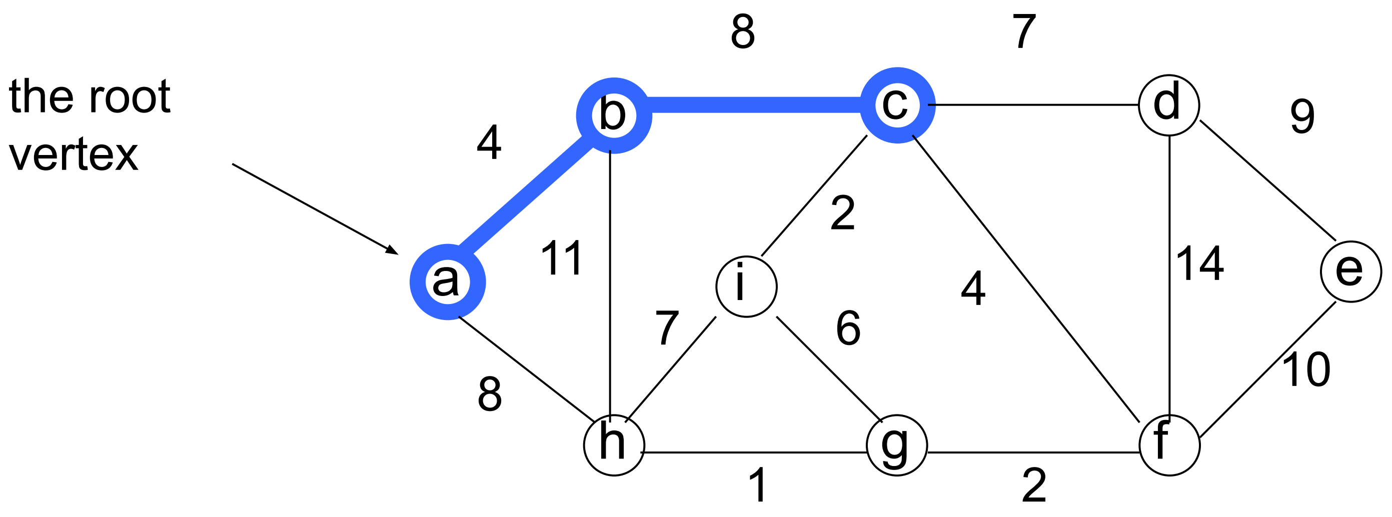
the root vertex




**Important:** Update Key[v] only if T[v]==0

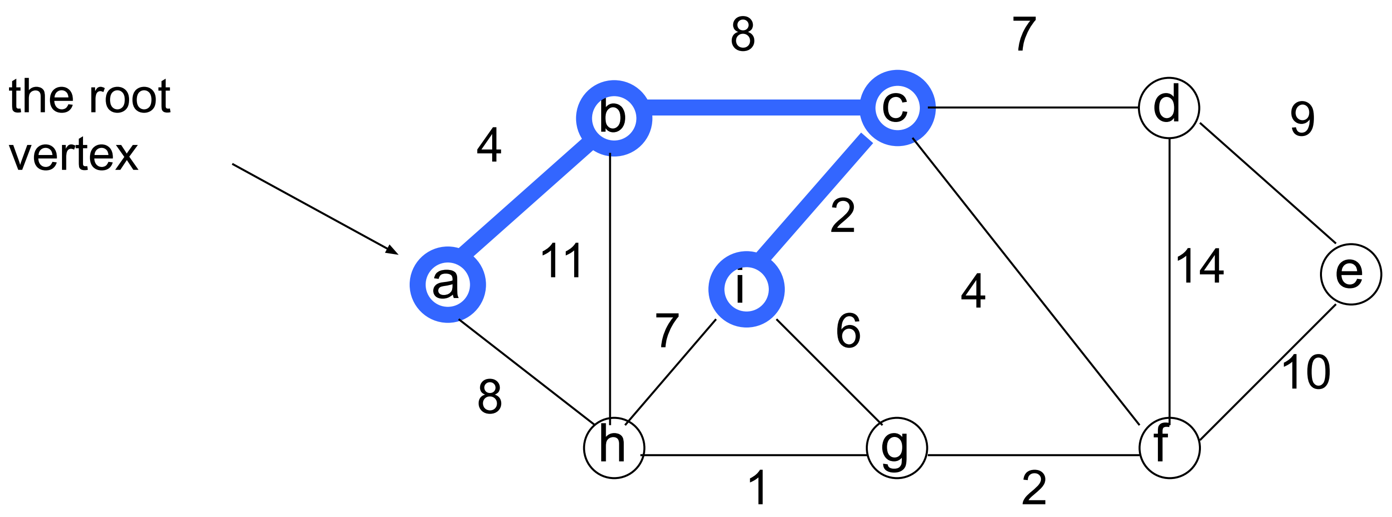
| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | **0** | 0 | 0 | 0 | 0 | 0 | 0 |
| Key | 0 | 4 | **8** | - | - | - | - | 8 | - |
| π | -1 | a | **b** | - | - | - | - | a | - |

# The execution of Prim's algorithm(moderate part)



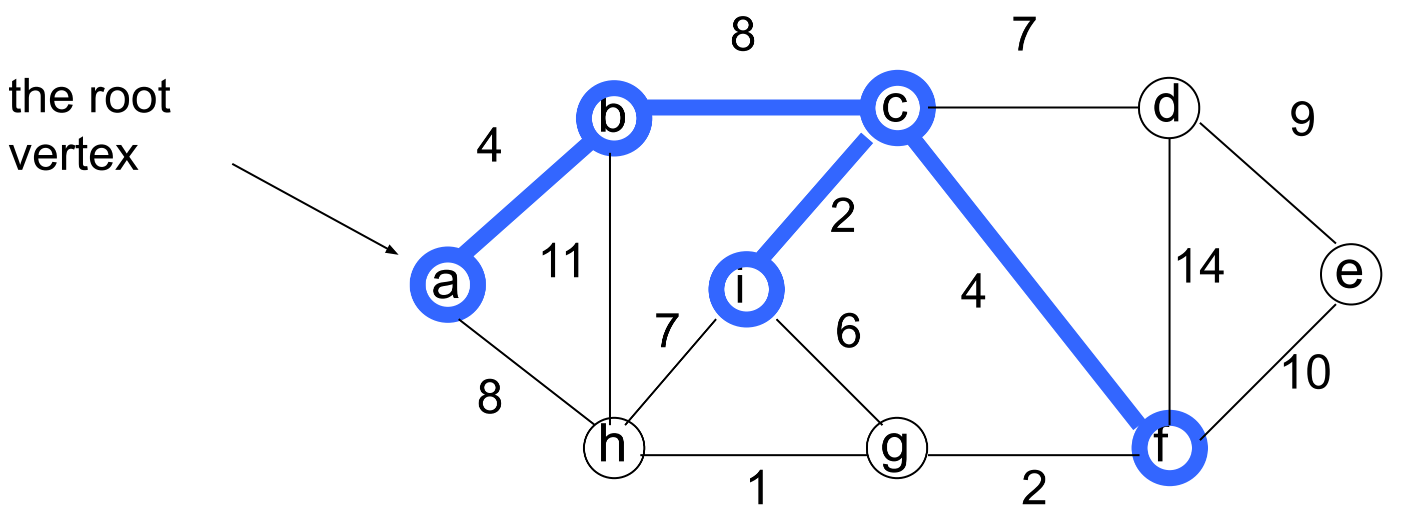

| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | **0** |
| Key | 0 | 4 | 8 | 7 | - | 4 | - | 8 | **2** |
| π | -1 | a | b | c | - | c | - | a | **c** |

# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | 0 | 0 | **0** | 0 | 0 | 1 |
| Key | 0 | 4 | 8 | 7 | - | **4** | 6 | 7 | 2 |
| π | -1 | a | b | c | - | **c** | i | i | c |

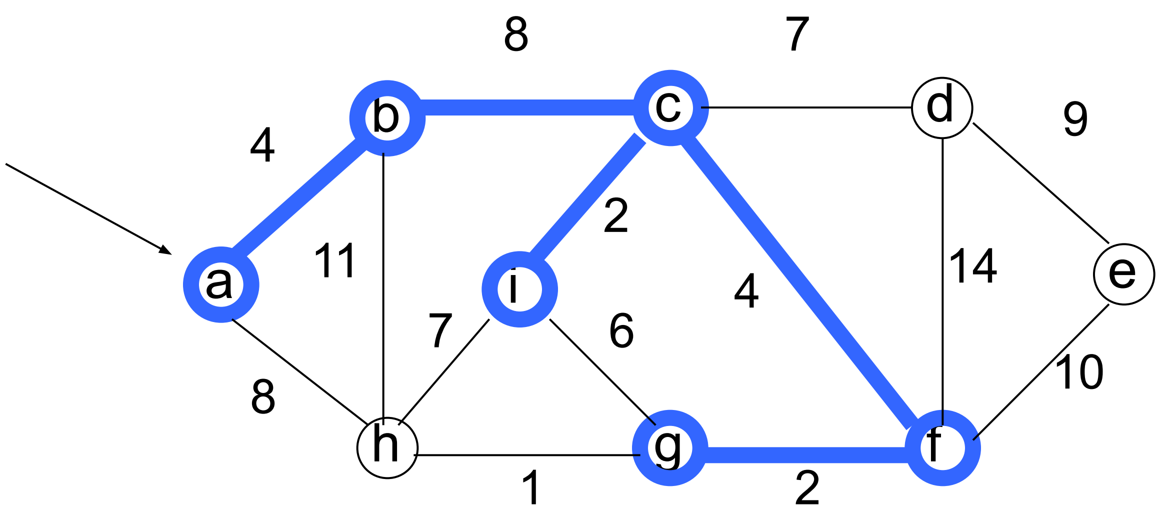
# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | 0 | 0 | **1** | **0** | 0 | 1 |
| Key | 0 | 4 | 8 | 7 | 10 | **4** | **2** | 7 | 2 |
| π | -1 | a | b | c | f | **c** | **f** | i | c |

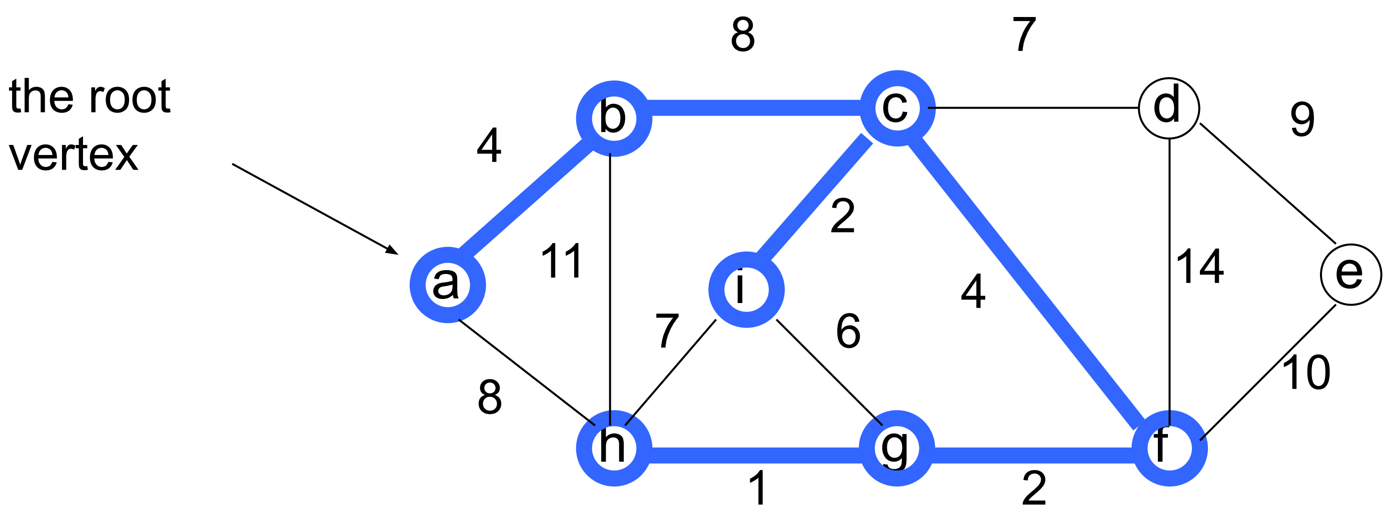
# The execution of Prim's algorithm(moderate part)

the root vertex



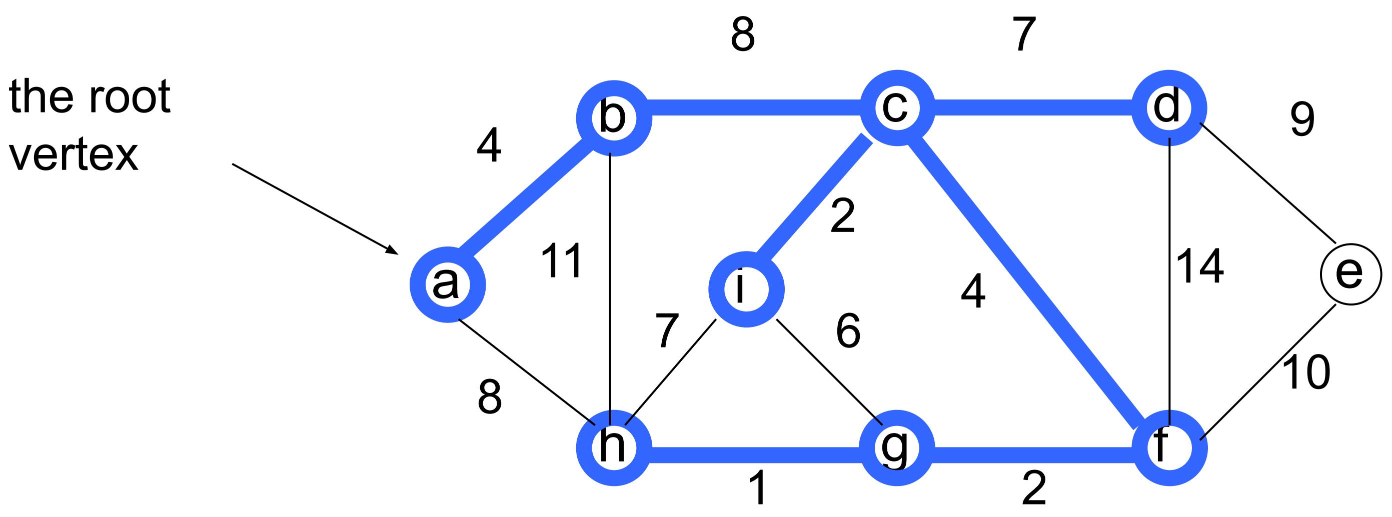

| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | 0 | 0 | 1 | **1** | **0** | 1 |
| Key | 0 | 4 | 8 | 7 | 10 | 4 | **2** | **1** | 2 |
| π | -1 | a | b | c | f | c | **f** | **g** | c |

# The execution of Prim's algorithm(moderate part)



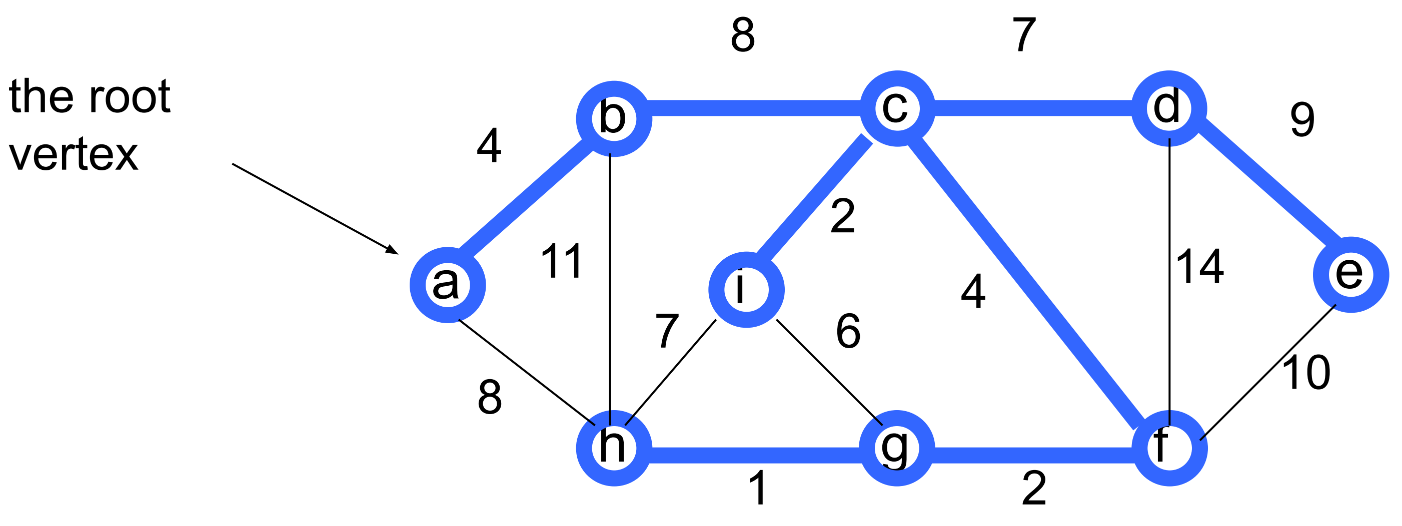

| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | **0** | 0 | 1 | 1 | **1** | 1 |
| Key | 0 | 4 | 8 | **7** | 10 | 4 | 2 | **1** | 2 |
| π | -1 | a | b | **c** | f | c | f | **g** | c |

# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | **1** | **0** | 1 | 1 | 1 | 1 |
| Key | 0 | 4 | 8 | **7** | **9** | 4 | 2 | 1 | 2 |
| π | -1 | a | b | **c** | **d** | c | f | g | c |

# The execution of Prim's algorithm(moderate part)




| V | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| T | 1 | 1 | 1 | 1 | **1** | 1 | 1 | 1 | 1 |
| Key | 0 | 4 | 8 | 7 | **9** | 4 | 2 | 1 | 2 |
| π | -1 | a | b | c | **d** | c | f | g | c |

# Complexity: Prim Algorithm

```
MST-Prim(G,w,r)
Q ← V[G]  // Q - vertices out of T
for each u ∈ Q
   key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅ do
   u ← ExtractMin(Q)   // making u part of T
      for each v ∈ Adj[u] do
         if v ∈ Q and w(u,v) < key[v] then
            π[v] ← u
            key[v] ← w(u,v)
```

O(V)

O(V)

Heap: O(lgV)

Overall: O(E)

Decrease Key: O(lgV)

Overall complexity: O($V$)+O($V$ lg $V$+$E$ lg $V$) = O($E$ lg $V$)

# Overall Complexity Analysis

- $O(V^2)$
  - When we don't use heap
  - To find the minimum element, we traverse the "KEY" array from beginning to end
  - We use adjacency matrix to update KEY.
- $O(E \log V)$
  - When min-heap is used to find the minimum element from "KEY".
- $O(E + V \log V)$
  - When fibonacci heap is used to find the minimum element from "KEY".

# Kruskal's Algorithm

- A MST can be grown from a forest of spanning trees by adding the smallest edge connecting two spanning trees.

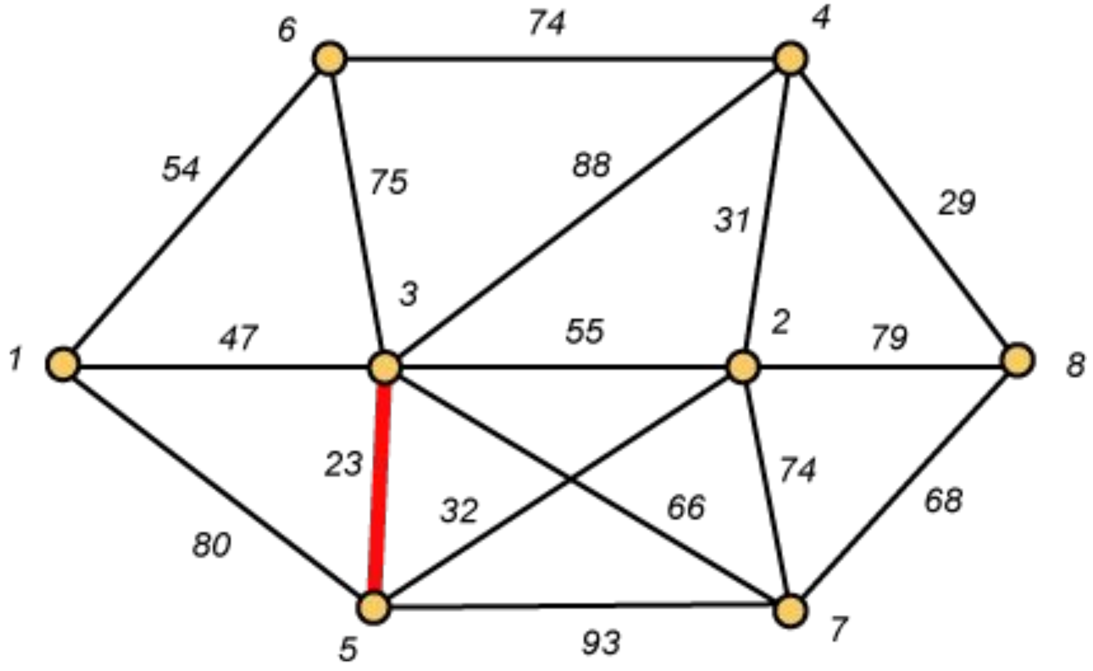
# Some Definition

- Cut:
  - Partition of V. Ex: (S, V-S)
- Cross:
  - Edge (u,v) crosses the cut (S, V-S) if one of its endpoints is in S and the other is in V-S.
- Light edge:
  - An edge crossing a cut if its weight is the minimum of any edge crossing the cut.

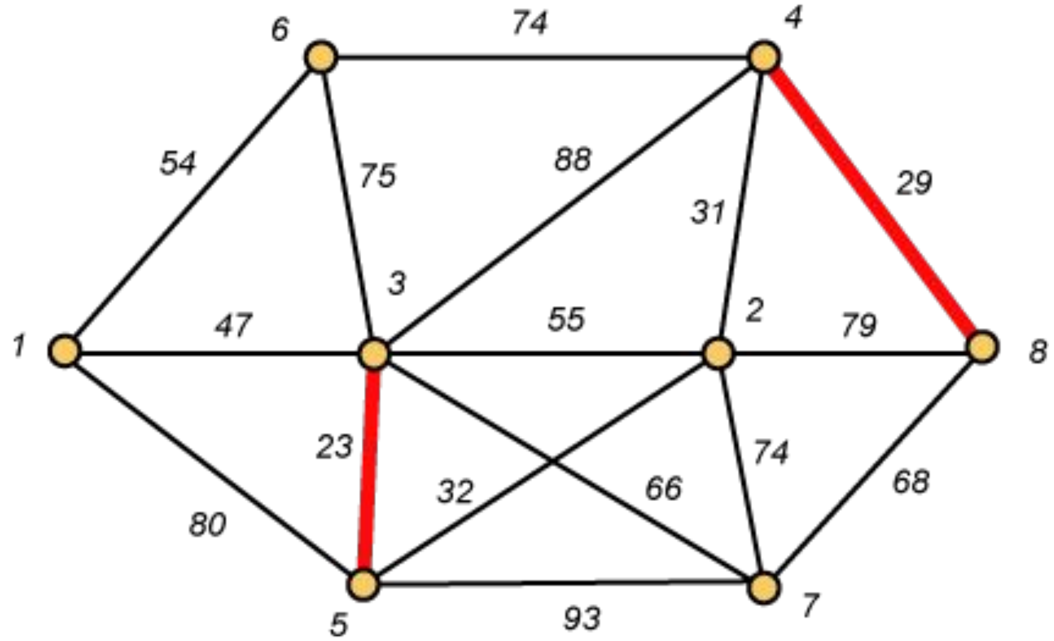
# Kruskal's Algorithm

- Edge based algorithm
- Add the edges one at a time, in increasing weight order
- The algorithm maintains *A* – a **forest of trees**. An edge is accepted it if connects vertices of distinct trees
- We need a data structure that maintains a partition, i.e.,a collection of disjoint sets
  - MakeSet(S,x)
  - Union($S_i$,$S_j$)
  - FindSet(S, x)

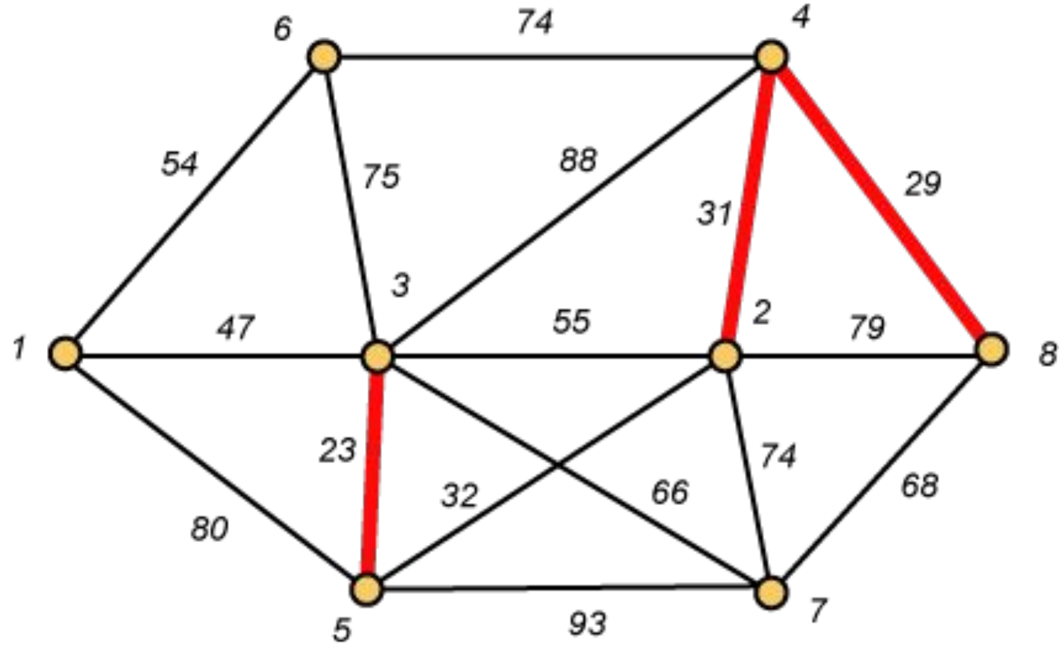
# Kruskal – Step 1



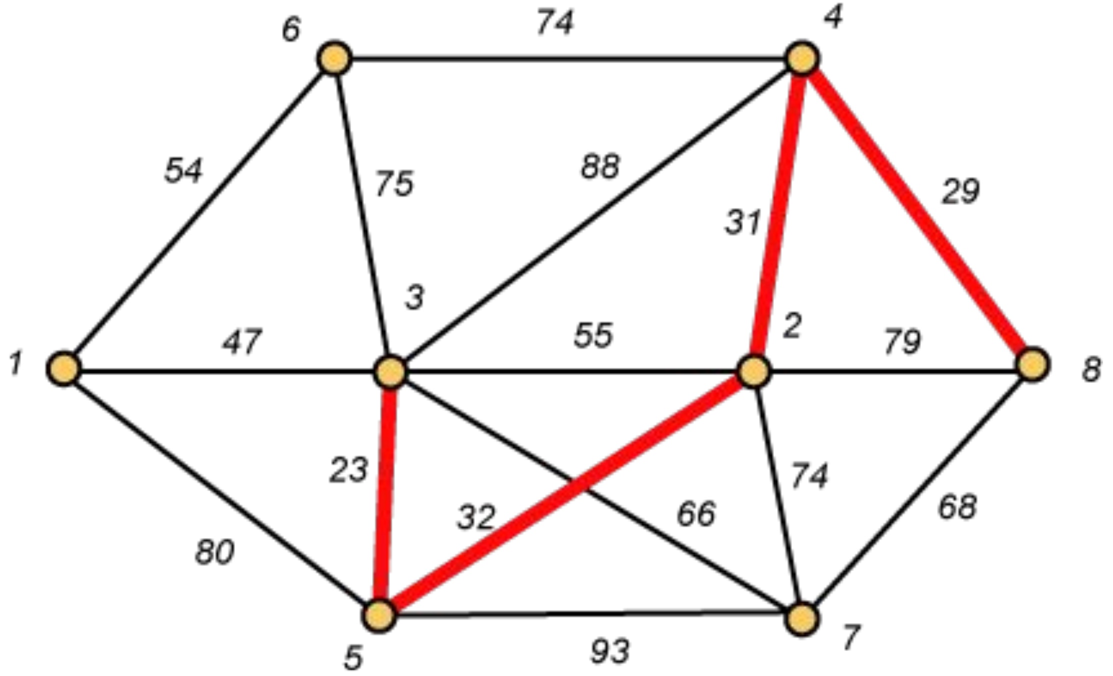
# Kruskal – Step 2



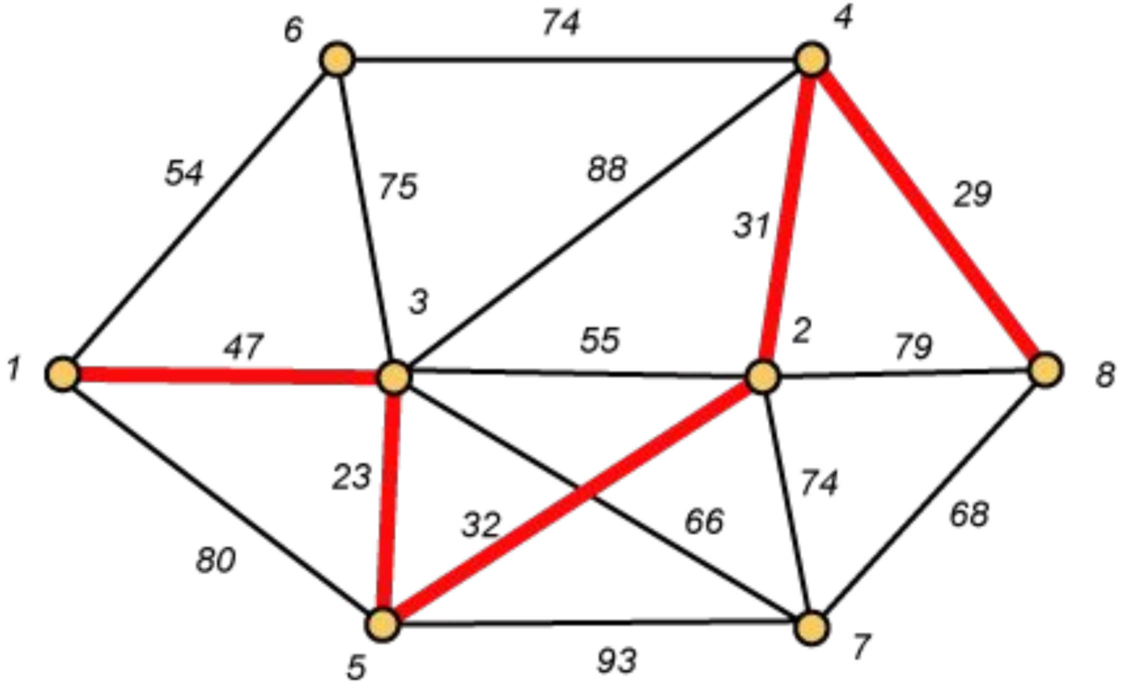
# Kruskal – Step 3



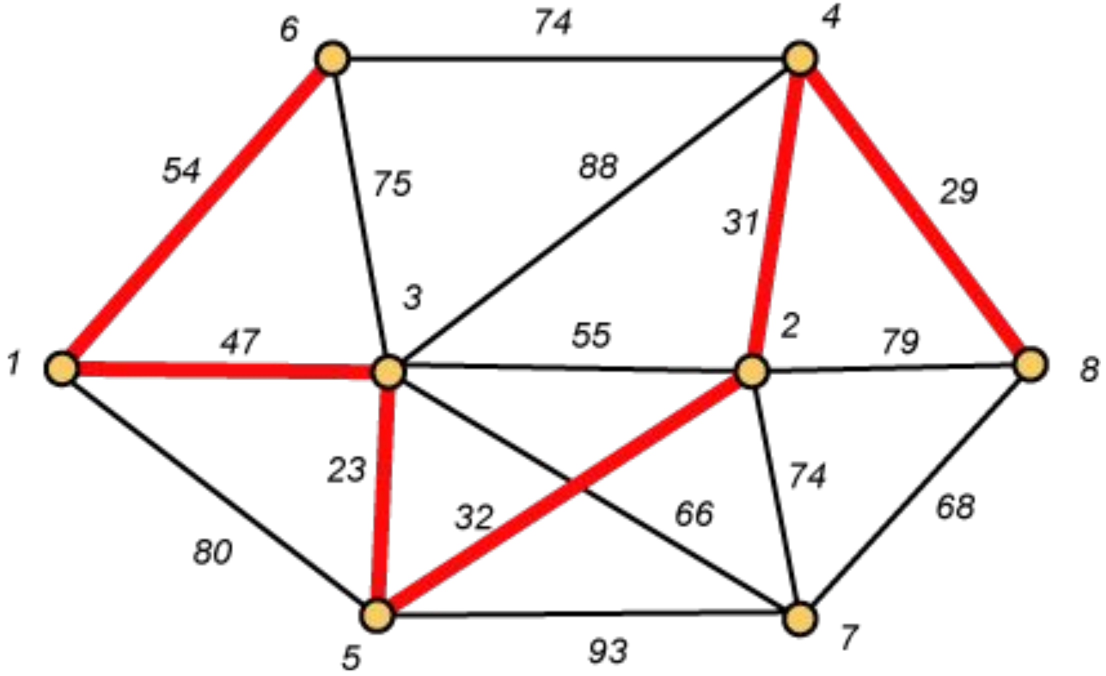
# Kruskal – Step 4



# Kruskal – Step 5



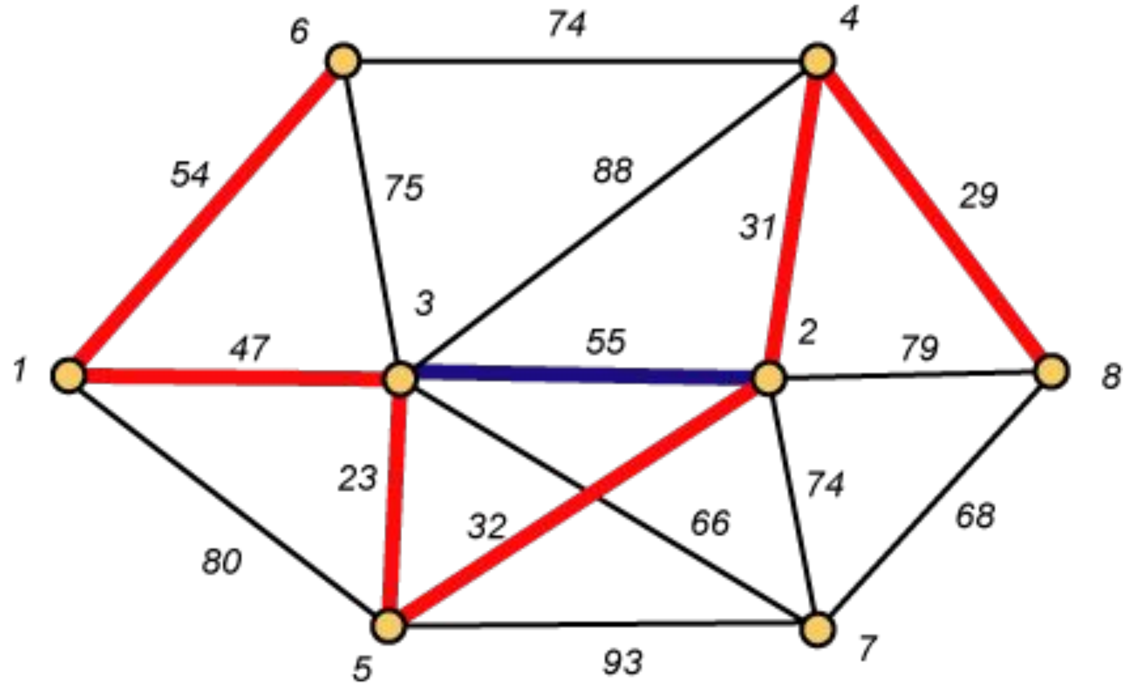
# Kruskal – Step 6



# Why Avoiding Cycles Matters

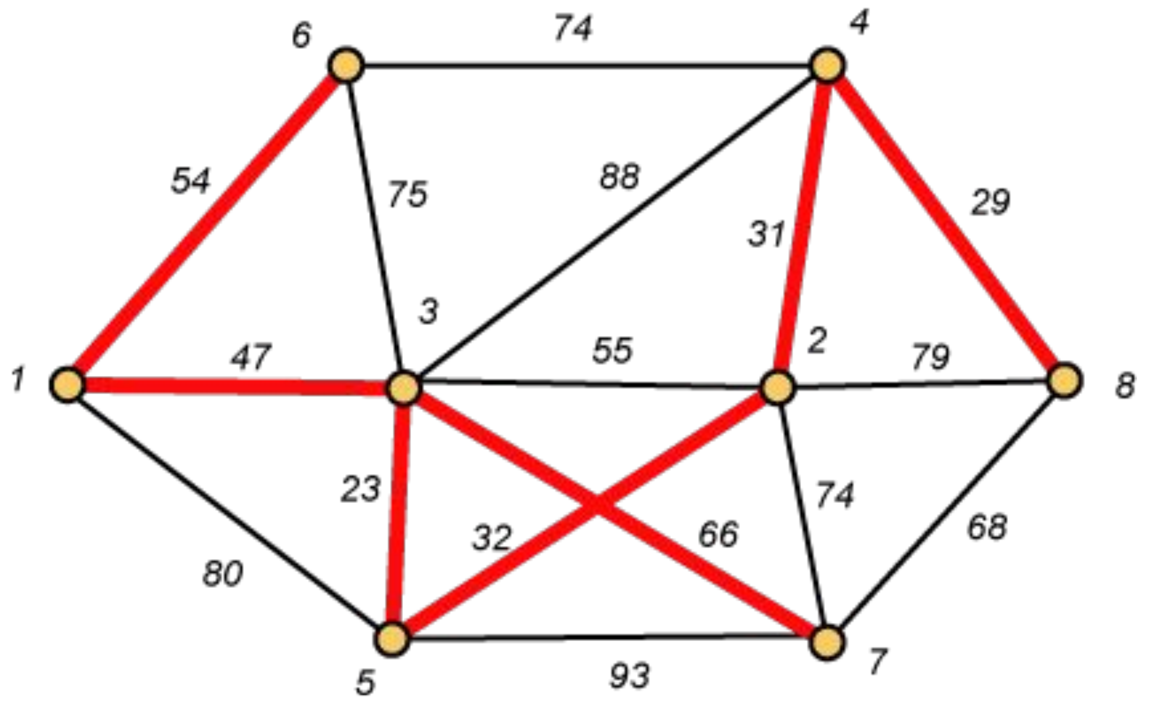
Up to this point, we have simply taken the edges in order of their weight. But now we will have to reject an edge since it forms a cycle when added to those already chosen.

# Forms a Cycle




So we cannot take the blue edge having weight 55.

# Kruskal – Step 7 *DONE!!*




Weight (T) = 23 + 29 + 31 + 32 + 47 + 54 + 66 = **282**

# Kruskal's Algorithm

- The algorithm adds the cheapest edge that connects two trees of the forest

```
MST-Kruskal(G,w)
A ← ∅
for each vertex v ∈ V[G] do                                         O(V)
    Make-Set(v)
sort the edges of E by non-decreasing weight w                       O(ElogE)
for each edge (u,v) ∈ E, in order by non-decreasing weight do        O(E)
    if Find-Set(u) ≠ Find-Set(v) then                                O(V)
        A ← A ∪ {(u,v)}
        Union(u,v)
return A
```

**Overall Complexity: O(VE)**

# Textbooks & Web References

- Text Book (Chapter 23)
- www.geeksforgeeks.org

Thank you

&

Any question?